DOCUMENT RESUME

ED 267 908 PS 015 707

AUTHOR Mandinach, Ellen B.
TITLE Aspects of Programming Courses That Foster Problem Solving.
INSTITUTION Educational Testing Service, Princeton, N.J.
PUB DATE Jan 86
NOTE 32p.; Document contains light type. Paper presented at the Annual Meeting of the American Educational Research Association (70th, San Francisco, CA, April 16-20, 1986).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Academic Achievement; Case Studies; Cognitive Processes; *Computer Software; Elementary Secondary Education; *Learning Strategies; *Middle Schools; *Problem Solving; *Programing; Teacher Characteristics; Teaching Methods

ABSTRACT

This study examined the relationship between type of computer programming instruction and successful performance by middle school students. It is argued that characterizing the aspects of programming courses that foster generalizable problem solving skills and domain-specific knowledge is helpful in assessing the potential of computer learning environments. Similarly, examining the achievements of successful programmers, their background characteristics, and the learning experiences that lead to expertise in programming is also helpful. Two exemplary programming courses from the Assessing the Cognitive Consequences of Computer Environments for Learning (ACCCEL) Project were examined in case study format. The teachers, curriculum materials, and instructional procedures were compared and characterized. The teachers' classroom procedures and instructional content were examined to identify critical influences on learning. In conclusion, results indicated that (1) the talented students were the best programmers in the ACCCEL sample and (2) good classroom instruction was an important influence on the knowledge and skills displayed by the talented students. (DST)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
NATIONAL INSTITUTE OF EDUCATION
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
X This document has been reproduced as received from the person or organization originating it.
☐ Minor changes have been made to improve reproduction quality.
• Points of view or opinions stated in this document do not necessarily represent official NIE position or policy.

ED267908

Aspects of Programming Courses that Foster Problem Solving

Ellen B. Mandinach

Educational Testing Service, Princeton, New Jersey

January, 1986

Paper presented at the annual meeting of the American Educational Research Association, San Francisco, April, 1986.

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Ellen Mandinach
TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

PS 015707

BEST COPY AVAILABLE

Abstract

To understand the factors leading to success in programming courses it is helpful to examine both the background characteristics of successful students and the computer programming instruction they received. Students often receive their first formal instruction in computer programming at the middle school level. Few students, however, acquire extensive computer experience outside of the classroom. The present study examined the relationship between the form of instruction and performance of successful students. Two exemplary programming courses were examined in case study format. The teachers, curriculum materials, and instructional procedures were compared and characterized. The teachers' classroom procedures and instructional content were examined to identify critical influences on learning. These classroom features were contrasted to determine how differences in the courses led to readily differentiated cognitive outcomes.

Aspects of Programming Courses that Foster Problem Solving

To assess the potential of computer learning environments, it is helpful to characterize the aspects of programming courses that foster generalizable problem solving skill as well as domain-specific knowledge. It also is helpful to examine the achievements of successful programmers, their background characteristics, and the learning experiences that lead to expertise in programming. A central motivation for the present study was to examine the skills of students who are the most successful in middle school programming classes and to determine the factors likely to accompany such success. Of particular interest was the identification and examination of instructional characteristics that foster the acquisition of skill and knowledge.

The study was part of a coordinated set of investigations of the cognitive consequences of middle school programming instruction. These studies were conducted by the Assessing the Cognitive Consequences of Computer Environments for Learning (ACCCEL) Project, funded by the National Institute of Education (Dalbey & Linn, in press a, b; Linn & Dalbey, in press; Mandinach, 1984, in press; Mandinach & Fisher, 1985; Mandinach & Linn, in press; Stein & Linn, 1985).

The avowed purpose of most programming courses is to teach reasoning or problem solving as well as to teach programming. However, results indicated that students in introductory courses generally made little progress in learning to program and, therefore were unlikely to gain general problem solving skills. Only a small percent of students in these courses learned how to solve problems in programming (Dalbey & Linn, in press a; Linn & Dalbey, in press). An important objective of the present study was to characterize not only the problem solving skills acquired by students who made the most progress, but also to identify the instructional factors that contributed to these students' progress.

To assess these skills it is useful to determine the sequence of events that leads to problem solving skills. The ACCCEL Project examined the behavior of expert programmers (c.f., Atwood & Ramsey, 1978; Kurland, Mawby, & Cahir, 1984), observed students and experts solving programming problems, and reviewed the contents of outstanding programming texts to devise what is called a chain of cognitive accomplishments that culminates in problem solving skill. This chain, described below, serves as a guide for determining how much progress students have made (Linn, 1985).

It also is helpful to examine the aspects of programming courses that are likely to foster the acquisition of skills along the chain of cognitive accomplishments. In the present study we considered the influences of instructional provisions, curriculum materials, and teacher behaviors that made programming instruction more or less effective

Chain of Cognitive Accomplishments

The ACCCEL Project identified a chain of cognitive accomplishments that culminates in the higher cognitive skills frequently thought to result from computer learning (Linn, 1985; Linn & Dalbey, in press; Mandinach, in press). This chain specifies how programming experiences can be linked to produce general problem solving skills. Programming instruction was chosen to exemplify this chain because it is a common and demanding use of the computer learning environment. A similar chain could be identified for software (Lockheed & Mandinach, 1985; Mandinach & Fisher, 1984). The chain is described briefly here (see Linn, 1985 for details).

Language features. The first link in the chain consists of knowledge of the features of the computer language. Features are the non-decomposable commands such as PRINT, GOTO, or IF...THEN in BASIC. Such knowledge is necessary to use the language but is not sufficient to foster higher cognitive skills.

Design features. The second link in the chain consists of techniques for combining language features to solve programming problems, and can be separated into templates and procedural skills. Templates are stereotypic patterns of code, longer than a single line, that perform reasonably complex functions. For example, templates for sorting, for reading input, or for computing the mean of a set of numbers are often taught. If students are taught templates for component tasks they can compose more sophisticated and powerful programs than if they have to invent the templates themselves. Templates also encourage effective use of programming language.

Procedural skills, such as planning, testing, and reformulating, are used to invent and refine programs. Although expert programmers use these skills constantly, novices sometimes fail to see their importance. For example, experts spend a reasonable amount of time planning their problem solutions, whereas novices often go directly to the computer without planning.

Problem solving skills. The third link in the chain consists of techniques for using generalized templates and procedural skills for one language when learning a new formal system. Such a formal system could be a new programming language, a database management system, or a computer controlled monitoring device. Of particular interest among problem solving skills are what could be called autonomous learning skills--skills needed to manage one's own mastery of new formal systems. They are autonomous in the sense that learners assume responsibility for their own learning (Corno & Mandinach, 1983; Glaser, 1984). Autonomous learning skills include the ability to: (a) interrogate a new system; (b) get useful information from teachers, experts, or consultants; and (c) use reference materials effectively.

Characteristics of Instruction

Instructional methods require activation of relevant cognitive processes. Cognitive demands, support, and characteristics that underlie an instructional method influence how, why, and to what extent students profit from specific treatments. Tobias (1982) defines instructional support as "assistance given the learner by way of organizing the instructional content, maintaining student attention, eliciting responses, providing feedback on responses, and so on" (p. 5). As Cronbach and Snow (1977) note, individuals differ with respect to how they profit from instruction and learn most efficiently. Learners with different aptitude profiles learn better under different instructional methods.

The directness or explicitness of instruction has been found to influence learning outcomes. Direct, explicit instruction benefits novices, young children, and students of low-ability, but is not highly appropriate for all learners (e.g., Doyle, 1983; Shulman & Keislar, 1966). Indirect instruction may provide abler students with practice in higher-order cognitive skills. The "unstructuredness" of the instruction forces learners to respond actively in comprehending and building their own knowledge structure of the materials.

All learning involves both implicit and explicit knowledge, yet individuals differ in the degree to which they actively process such knowledge. Instruction can make forms of knowledge more or less salient by placing on or removing information-processing burdens from the learner (Snow, 1977). Treatments that relieve information-processing burdens benefit low-ability learners and apparently interfere with pre-existing knowledge structures of high-ability students (Egan & Greeno, 1973; Mayer & Greeno, 1972). Conversely, high- profit more than low-ability students from treatments that make heavy processing demands. In general, highly supportive instruction benefits low-ability learners by relieving many of the processing demands, whereas such instruction can be detrimental to high-ability students. High-ability students tend to perform better when given less instructional support because they can impose previously successful procedures on the learning task (Tobias, 1982).

Another instructional characteristic that is likely to influence learning outcomes is the use of examples. However, there is a tradeoff between the explicitness of the rules from which general propositions are to be induced and the risk of errors. The risk is of inducing incorrect rules when examples are presented before the rules. Rule-example sequences decrease information processing burdens and are "instruction-driven". Example-rule sequences are "student-driven" and allow learners to impose their own cognitive structures on the instructional material. Students may be unable to induce the necessary information from the example-rule sequences. However, Anderson's (1967) review indicates that the rule-example sequences tend to

produce faster learning, fewer errors, and better retention than example-rule sequences. The appropriateness of this generalization across students with different aptitude profiles and cognitive styles has yet to be confirmed.

Several cognitive skills are required in learning from examples. Students can induce knowledge by abstracting representations of structured patterns from sequences of examples and by identifying common properties among exemplars, using processes called matching and interference (Hayes-Roth & McDermott, 1976). Students learn by comparing, recognizing general rules, discriminating relevant from irrelevant properties, and practicing rule applications provided in instructional examples (Hayes-Roth, 1978). Positive and negative examples help students to discriminate between relevant and irrelevant facets of the stimuli and learn to distinguish when and when not rules are applicable across situations (Ali, 1981).

The above mentioned instructional variations are likely to affect how students learn computer programming. However, these variables are by no means the only factors in computer programming instruction. Research in instructional methods is too vast to examine here. The ensuing study is confined to examining the effects of instructional explicitness, learning by examples, and describing the instructional provisions, curriculum materials, and teacher behaviors that characterize effective and less effective instruction in computer programming.

Research Questions

The present study addressed two general questions. A first question focused on student achievement. How far along the chain of cognitive accomplishments are the most successful middle school programmers? This question was addressed more fully elsewhere (Mandinach & Linn, 1985, in press) and discussed briefly here. Instead, the focus in the present analysis is on classroom-level characteristics and instructional variations. What are the specific instructional procedures, curriculum materials, and teacher behaviors that lead to increases in the acquisition of computer programming skills and knowledge?

Method

Sample Selection

Talented Student Selection

The subject pool for the main ACCCEL sample consisted of a total of 2400 students who were enrolled in six middle schools in the Bay Area (see Mandinach & Fisher, 1985 for details). Approximately 600 of the students enrolled in programming courses participated in the main ACCCEL study from which we sought to select the top one percent of the students for the experienced programmers substudy (hereafter referred to as the EPS).

A two step process was used to identify these top students (see Mandinach & Linn, 1985, in press). First, nominations and recommendations were solicited, then the most talented were selected on the basis of a programming test. Based on performance of the top students in the schools offering the most rigorous instruction in programming, we asked teachers, students, and administrators to nominate students who could, on their own, write simple programs. Our focus was on the three most demanding courses in the main ACCCEL sample (see Mandinach & Fisher, 1985 for details). A total of 98 students, referred to as experienced programmers, were nominated; 9 from one school, 32 from another, and 56 from the third. There were 53 males and 45 females. The average age of the sample was 13.1 years.

To select the top students (referred to as talented programmers), we administered a programming test to the experienced programmers. The talented subsample consisted of 24 students; 9 were male and 15 were female. Four students came from one school, 11 from another, and nine from the third.

Procedures

The original intention of the EPS was to examine student-level performance and background variables. However, it became increasingly clear that the instruction received by the EPS students was distinct from and superior to other classrooms procedures observed in the ACCCEL studies. Thus, the present focus is on the instructional characteristics of the classrooms from which these talented and experienced students came. Case studies of these classroom environments are described. However, a case study of the instruction at only these schools would neglect the background information provided by the two larger studies in which they were examined. Consequently, we will characterize briefly the instruction found in ACCCEL's main study (Linn, 1985; Linn & Dalbey, in press; Mandinach & Fisher, 1985) as well as that found in the EPS (Mandinach & Linn, 1985, in press), describe the main student-level achievement results from the substudy, and discuss in detail the instructional characteristics that made the classrooms from which the talented and experienced students were drawn exemplary.

Data collection in the main study. The ACCCEL Project carried out several substudies of programming skills within the framework of a multipurpose design (Linn, Fisher, Dalbey, Mandinach, Burbules, & Reese, 1983). Three instructional procedures were studied in the six middle schools. First, we examined the naturally occurring procedures in classrooms that provided typical instruction. Second, we focused on instruction in "exemplary" classrooms. Third, we supplemented extant instruction in typical classes with programming augmentations. Achievement in programming was assessed, as well as the relation between skill acquisition and student background characteristics (e.g., ability, access to computers, previous computer experience, interest, and gender).

Data collection in the EPS substudy. The talented students, as identified above, were given a series of on-line and off-line tests to assess their knowledge and skills in BASIC, including a fairly demanding program to be written over a three-week time period. To assess their level of problem solving skills, the students also were asked to write three programs in a language new to them. Again, analyses focused on both achievement in programming and the identification of student background characteristics that related to the performance measures. Central to the present analysis, instruction, classroom procedures, and teacher behaviors also were characterized and related to skill acquisition.

Results

Site Descriptions

Characteristics of schools that participated in the ACCCEL Project are outlined in Table 1. Students who participated in the EPS were selected from those at Sites 2, 5, and 6.

Typical and exemplary classrooms. Four of ACCCEL's schools provided what might be referred to as typical instruction. The schools were typical in that classes consisted of 12 weeks of instruction, had at least eight computers available, and teachers had participated in one or more in-service computer workshops prior to giving the class. The three classes at the most demanding of the typical schools served as one site for examination in the EPS. This school, referred to as School 2, offered three introductory computer literacy courses in which 84 students were enrolled.

The remaining two schools in the main study offered what is referred to as exemplary instruction. Students were taught explicitly how to design programs. The emphasis was on structured program design. Also notable about these sites was that the teachers were more experienced and the students were, on average, higher in general ability than those in the typical schools.

Classroom context. Classtime access to the computers varied across the sites and was a function of the number of students and computers per class. Students at Sites 2, 4, and 6 had approximately 20 minutes on-line time per day. Students at Sites 3 and 5 had only 10 minutes access, while those at Site 1 had about 15 minutes per class period. Schools generally allowed occasional out-of-class access to the computers, with Site 3 providing 45 minutes per day during lunch, Site 5 allowing two hours every Monday on a sign-up basis, and Site 6 providing access to the computers before and after school as well as at lunch.

Teachers. Of the six teachers, only one had formal training in computer science and had experience teaching in the subject area. Most teachers had taught programming for 1.5 to 2.5 years and were self-taught. In general, the teachers were not

experienced in computer science, but applied pedagogical techniques from other content areas to their programming classes.

Curriculum. Course content, level of difficulty, and instructional materials differed across sites. Sites 1 and 2 used informal and self-paced classroom procedures, whereas all other schools gave daily assignments and projects. Two of the schools used teacher-generated materials, in contrast to textbooks at the remaining sites. Only at Site 5 was the textbook a critical part of the curriculum, with the teacher structuring the entire course around the book. The curriculum at Site 6 was judged to be high in difficulty and rigor. Course difficulty was low at Site 3, slightly more demanding at Sites 1 and 2, and considerably more rigorous at Sites 4 and 5.

Instruction at the EPS Sites

Site 2. A "Computer Literacy" course was offered as an elective, with no prerequisites to seventh and eighth grade students at Site 2. The 1983-84 academic year was the third time that BASIC programming was offered at the school. There were 84 students enrolled in the three sections of the course. These students primarily were seventh-graders (61%) and male (68%).

Instruction was given daily for 12 weeks in 45-minute periods. The course consisted of ACCCEL's three-week programming introductions (see Dalbey & Linn, in press) and a curriculum focused on the textbook, Computer Literacy (Luehrmann & Peckham, 1983). Supplemental materials included programming exercises and various pieces of software (e.g., Algebra Arcade). Instruction was self-paced, moderately difficult, and emphasized basic syntax and language features.

Classroom procedures consisted of 5 to 10 minutes of discussion on the daily assignment, followed by on-line and off-line work on various computer activities. Students had about 20 minutes of on-line time per class during which they worked either individually or with partners.

Although the curriculum centered around the textbook, instruction necessarily varied from the implementation recommended in the Teacher's Guide. The teacher chose to use a self-paced format because there was a wide range of ability levels in the classes (i.e., gifted to special education). Several students were unable to discipline themselves in such a self-paced course, and therefore were failing. The teacher structured the learning environment with timelines for each assignment for these students.

Each class was divided into two groups: one group worked off-line, the other on-line. Each group had 20 minutes of on-line time and then was rotated off to seatwork. Students used notebooks for homework assignments, worksheets, and projects.

These notebooks were collected and examined by the teacher at regular intervals.

Students progressed through the 67 lessons in the textbook at their own pace. A classroom chart was used to record the number of lessons each student completed. The teacher monitored this chart and provided explicit timelines for students who needed additional assistance in planning their time.

The teacher spent most of each class monitoring the students who were working off-line. On-line students generally worked independently without teacher supervision. Thus, progress was dependent on the self-discipline of individual students. Few students exhibited the needed discipline, and therefore did not progress far into the lessons. Many of these unsupervised students used the on-line time to play computer games and manipulate files, rather than learn the new skills outlined in the lessons. A few students, however, exhibited autonomous behavior and made productive use of their time.

Site 5. An introductory computer programming course was offered as a two-semester elective for eighth grade students at Site 5. There were no prerequisites for enrollment in this course. Approximately half of the eighth grade class elected to take the class, two-thirds of whom were male. Students at this site predominantly were high in ability; one-quarter of the school's 634 students were identified as gifted.

Six sections of programming were offered. Two of the classes (each with about 30 students) participated in the ACCCEL Naturally Occurring Explicitness substudy. Many of these students also participated in the EPS. The classes met daily for 45 minute periods over 32 weeks. Students had about 10 minutes of computer access per class, as well as the use of an open laboratory every Monday afternoon (two hours on a sign-up basis). From eight to 20 students made use of this time each week.

Instruction at Site 5 was highly structured and closely followed the prescribed sequence of activities outlined in the textbook. The text selected was Computer Literacy (Luehrmann & Peckham, 1983). The teacher knew this book intimately as she had helped to field-test it prior to publication.

Students completed the 67 lessons in the text which ranged from simple language features to more complex design skills. The teacher emphasized structured programming techniques and, in particular, the use of templates (posters of useful templates were displayed around the classroom). Following the textbook sessions, the teacher developed a curriculum designed to teach arrays, nested loops, and read data statements. The teacher-selected materials provided exercises in coding and reformulation for each concept. This instruction was supplemented with extra credit projects that students could elect to do. Programming assignments

generally were challenging and required more complex design and procedural skills than exercises found at other ACCCEL sites.

The year-long classes were rigorous, structured, and well-organized. All six sections of programming were conducted in an identical manner. Each class consisted of about 10 minutes of teacher-led discussion, recall, and questions that focused on the previous night's homework. When the teacher was satisfied with the classroom recitation, students then were allowed to begin work on their programming assignments. Half of the students worked on off-line activities, while the remainder of the class went on-line in dyads. The two groups exchanged places halfway though the period. The procedures used in the class were highly systematized, with the students knowing where they were to be at any given time and on what activities they were to work. The teacher maintained strict discipline.

The curriculum's level of difficulty was moderate to high. The focus of instruction was on structured programming, specific language features, and templates. The teacher demanded well-structured code written in a specified format. Documentation, particularly with the use of subroutines, was required by the teacher. Such structure resulted in students writing very similar programs. Observers were able to distinguish programs written by Site 5's students from those produced at other schools simply by examining the programs' structure. Feedback provided by the teacher was direct. She almost always provided answers in an explicit manner or directed students to a reference or study aid from which appropriate code could be obtained (e.g., a template extracted from the textbook or another program). Students generally followed the lessons in the text and were not encouraged to invoke procedures that had not yet been covered in class. That is, the teacher preferred that students use less efficient, but previously taught concepts, rather than allowing students to attempt autonomously procedures they may have acquired elsewhere.

Many classroom resources were available to students at Site 5. In addition to the sets of instructions displayed on the bulletin boards, students used the text and its accompanying "home" disk to select appropriate extant code for reformulation into their own programs. Students were encouraged to select the resources that offered the most appropriate solutions to the given problem. One such resource often was "code bumming" from other students.

Because students worked in dyads at the computers, various types of peer interactions were encouraged and observed. Examples of positive interactions included peers who taught one another specific programming techniques, debated about how to write code for given problems, or discussed causes and solutions to particular bugs. However, not all of the peer interactions were positive. Working in dyads increased the likelihood that one student would do all the work, while the other sat idle. Different scenarios were observed: (a) one student typed the

code without prior thought or planning, while the other performed did most of the mental work; or (b) one student did all of the work, while the other exhibited inattentive and off-task behavior.

In general, most of the instruction at Site 5 closely mirrored the textbook's prescribed sequence of activities. Autonomous learning behavior was limited severely as the class was required to move through the curriculum together. For students with prior computer experience, the pace was too slow and they were not allowed to move ahead. However, upon completion of the text, students were given slightly more autonomy. Extra credit projects were attempted by students wishing to move ahead. Although students were not allowed to progress at their own pace, they did attempt to use existing classroom resources to write programs and find solutions to their bugs.

Site 6. A one semester computer programming course was offered at Site 6 to eighth grade students during both semesters of the 1983-84 academic year. Only 25 of the students who elected to take the course were allowed to enroll each semester. The minimal requirement for enrollment was a B+ in Algebra and English. Additionally, students were selected according to gender; that is, an attempt was made to have an equal number of males and females in each class. Both the first and second semester classes participated in ACCCEL's Naturally Occurring Explicitness substudy. Many students also were selected for the EPS.

Classes met daily for 45 minutes during a 12 week semester. Students had about 20 minutes of on-line time during class. The teacher also made the computer laboratory available before and after school, as well as at lunch (a total of about 15 extra hours of access time available per week). Each student was encouraged to make use of the computer availability.

It is important to note that the teacher at Site 6 differed from all other teachers that ACCCEL contacted throughout the two-year study. The teacher at this school was an excellent programmer, familiar with several programming languages, who also taught computer science at a local community college. Originally a mathematics teacher (and chairperson of the department), she had taken and continues to take computer courses to increase her knowledge of the content area. She stressed procedural knowledge, fostered autonomous learning, and exuded enthusiasm for the content area and concern for students. Her highly effective pedagogical style and classroom procedures were not unique to the programming classes, but generalized across subject areas (three math classes per semester).

It is also noteworthy that while most schools (the two above mentioned sites as well) generally used one type of microcomputer, the teacher at this site insisted that students become familiar with several types of computer equipment. The laboratory

contained a classroom set of Apples, but also was equipped with Pet (interconnected with a hard disk system), IBM, and Olivetti (IBM compatible) computers. Students were required to demonstrate their ability to use each type of machine.

The curriculum at Site 6 was more demanding and rigorous than at any other ACCCEL site. It was developed by the teacher and emphasized general problem solving skills as well as programming. The teacher used problem sets and instructional materials that she had collected over the years. Projects were selected and assigned from this library of curriculum materials. The teacher had not yet found a textbook with which she was satisfied and that complemented her focus on problem solving and procedural skills. She did, however, maintain a classroom library of different textbooks, computer magazines, and operating manuals which students used as reference materials.

Classes were structured so that students spent two days per week on-line and the remaining three in lectures. Students were expected to use the computers outside of classtime to complete their assignments. Lecture time decreased and was replaced by more on-line time as the course progressed. Observations indicated that the classes were well-organized and that students made productive use of both on-line and off-line time. Students who worked off-line planned their programs, prepared their assignments, and worked independently. They generally prepared elaborate written code and designs for their programs during this off-line time. The on-line students elaborated on the prepared designs, diagrams, and programs. Thus, there was little wasted time, both on- and off-line due to student preparation. The separation of on- and off-line time also helped students to plan their programs efficiently. Programs generally were well-documented and rarely contained spaghetti code. Although the assignments were the same for all students, the products indicated a variety of creative and innovative solutions.

Students completed weekly and bi-weekly programming projects throughout the semester. Upon completion of the semester, they were able to work with two dimensional arrays and string functions. Students also were expected to complete a major project by the end of the semester. This project helped to foster autonomous learning skills and completed the classroom structure which encouraged independent behavior on the part of the students. For example, there was a classroom rule that students must attempt to solve problems for 10 minutes before seeking help from the teacher. If after 10 minutes a solution was not found, the student then could ask the teacher for help.

The feedback given by the teacher was rarely in the form of a direct answer to a specific question. Rather, she forced the students to think about the problems and solve them on their own, providing subtle hints and possible directions to follow toward a solution. The teacher modeled appropriate planning and debugging behavior and problem solving skills in providing feedback to the

students. The feedback emphasized testing--students were told how to determine and isolate general types of errors. It stressed how to recognize a class of bugs, rather than how to solve a specific problem. Both the required 10 minute wait time, in conjunction with the indirect feedback forced students to develop and test hypotheses, autonomously debug their programs, and to seek answers independently. It was common to see a small group of students attempting to solve a problem before approaching the teacher for assistance. The teacher then would use the Socratic method with the group to help them isolate the bug. This instructional procedure also generalized to interactions among students who were collaborating on a project.

A good deal of peer interaction and help was observed in the classroom. Students provided helped to each other in a constructive manner and readily accepted such advice. Because only a select group of students were chosen to participate in this course, there was a high degree of incentive to excel. Those students with low performance were not selected as partners on projects.

On the whole, Site 6's programming course was unique in that it was taught more like a college- than a junior high school-level class. The course content was by far more demanding than other junior high school classes. But more importantly, the curriculum placed the responsibility for learning on the students. This responsibility, in turn, helped to foster not only content-specific knowledge, but also autonomous learning and general problem solving skills. The teacher demonstrated programming expertise, as well as excellent pedagogical techniques. This expertise, in conjunction with the amount of available computer access encouraged students apply problem solving skills, use existing resources to find solutions, be autonomous learners, and strive toward a high level of achievement.

Student-Level Achievement

The results from the programming assessments were used to establish how far along the chain of cognitive accomplishments the EPS students progressed. Results often are displayed by school to highlight the performance differences that reflect the role of instruction (see Mandinach & Fisher, 1985; Mandinach & Linn, 1985, in press for details).

ACCCEL Final Programming Assessment. Performance on the Final Programming Assessment was examined to compare the talented programmers to all students in the ACCCEL sample (Table 2). Talented students exhibited superior performance, achieving 90% correct (t (24)=2.30, $p < .05$). The experienced but not talented group (78%) performed slightly better than students at Sites 5 and 6 (72%), where most of the talented students were found. The subsamples performed at a much higher level than students in the total ACCCEL sample, who achieved only 38% correct.

On-Line Programming Assessment. Performance on the measure varied across individuals and schools (Tables 3 and 4). Programs were not readily interpretable, either in listing (58%) or on-line (44%) form. School 5's listings were most readable, reflecting the instruction which encouraged use of remarks and indenting to make code understandable. For the most part, and to the expressed dismay of two of the three teachers, the programs were not documented sufficiently to stand alone. Students at Site 6 included the most documentation in their programs. This finding again reflected an instructional emphasis on making programs user-friendly.

Performance on the use of control structures varied across students (range from 0 to 100%). Students either knew how to write a loop or subroutine or were completely lacking in such knowledge. On average, the programs were fairly well organized with logical and efficient code (70% on the structure index). Performance variations corresponded to the schools' instructional emphases. Students at Site 6 were considerably more adept at structuring their programs than were those from Site 2, where programs often contained numerous lines of redundant code.

A discriminant analysis was performed to determine if site differences were apparent from students' performance on the programming task. With the execption of the one outstanding programmer (see Mandinach & Linn, 1985 for details) at Site 2, it was possible to distinguish the students' school by examining the form of their programs. The discriminant analysis indicated that 77.27% of the students were correctly classifiable by school. In the analysis, the outstanding student at Site 2 was unique and assigned to Site 2. The other Site 2 students were incorrectly assigned to Site 6.

Long-Term Programming Assessment. Thirteen students elected to work on the Hurkle program and 11 on Bagels. Results indicated performance trends similar to those noted for the shorter programs (Tables 5 and 6). Readability improved slightly, presumably because there were no time constraints. Readers generally could understand the programs given the documentation and remarks. Several students, however, omitted directions to assist the user.

Scores on overall quality (62%) and use of control structures (66%) were similar to those for the shorter programs, suggesting that additional time was not a factor in determining the form of the program. Reflecting instructional differences, students at Site 2 were not as adept at subroutines and loops, compared to the other students. Considerable individual variation was found for adherence to specifications.

A discriminant analysis using the Long-Term Programming Assessment to assign students to groups classified almost as many students correctly (71%) by school as did the analysis using the On-line Programming Assessment. When taken in combination, the two tasks correctly classified 86% of the students by school. The

three lower scoring students at Site 2 were grouped together, whereas the outstanding student was classified with students at Site 5.

Spider World Assessment. The Spider World tasks assessed the problem solving link on the chain of cognitive accomplishments. The first task tested acquisition of language features. Students performed fairly well on this task, obtaining 72% of the possible points. The second task required the use of design skills and more complex language features (e.g., subroutines and loops). On average, students obtained 52% of the possible points. A wide range of performance was observed. Some students were able to apply the most complex procedures successfully, whereas others were unable to provide even structure for the exercise. Only 12 students attempted the final and most complex task (which was a logical extension of the second, requiring a combination of advanced procedural skills and templates). The select group of students who worked on the problem performed well, obtaining an average of 71% of the points.

Results indicated that in less than one hour, most of the talented students became familiar with the simulation, demonstrated their knowledge of the language features and control structures, and were able to plan efficient and well-structured programs. Students who worked with Spider World demonstrated the ability to learn a new formal system by writing fairly complex programs in a short period of time. With only minimal instructions, the students proceeded, without stopping to read manuals or ask questions. They explored the microworld freely, exhibiting autonomous learning and general problem solving skills. In contrast, other ACCCEL students made less progress in three weeks of formal classroom instruction. The talented students tested the system, at first by trial and error, but became increasingly systematic as their knowledge of the environment's constraints grew. They sought help from resources only when the on-line information was insufficient.

Discussion

Programming

Results indicated that the most successful programmers in middle school did not progress far along the chain of cognitive accomplishments for programming. Distinct performance variations across schools were noted, suggesting the strong influence of the instruction students received.

Chain of Cognitive Accomplishments

The talented students mastered the main features of BASIC and were beyond the language feature link on the chain of cognitive accomplishments. They autonomously could design programs of moderate complexity and use previously acquired templates to solve parts of the problems. They did not, however, have the templates

used by experts for certain common operations. The students had gained certain procedural skills, demonstrating some skill in planning their programs before going on-line. However, the programs were not always as efficient as they could have been. Students often used several lines of code when fewer lines would have performed the needed function.

A major deficit was in testing skill. Students failed to test their programs to see how they would respond to acceptable but unusual input and, as a result, turned in programs that behaved erratically. They were, however, adept at reformulation when their programs had syntax errors. Thus, students were making progress in acquiring procedural skills but had not yet mastered these skills.

Acquisition of problem solving skills was reflected in the students' ability in learning a new formal system. The talented students were much more successful on the Spider World problems in under one hour of exposure and in a discovery learning paradigm than other ACCCEL students were in three weeks of formal instruction (Dalbey & Linn, in press a). It is noteworthy that some students evidenced skills at the third link of the chain but had not yet mastered the design link.

Instruction

All but one of the students in the talented sample learned programming in school. Students were successful in mastering the instructional content, but did not go beyond what was taught explicitly. Schools with exemplary curricula produced the students farthest along the chain of cognitive accomplishments and the majority of the talented programmers. As Linn and Dalbey (in press) discuss, many factors contributed to the success of these exemplary classes. The teachers were experienced, the curricula well-developed and tested, the students generally of medium to high ability compared to the other schools, and computer access used effectively. This combination of factors was effective for teaching students BASIC.

Interestingly, even at these exemplary schools, students were only at the design link in the chain of cognitive accomplishments. They had not mastered procedural skills, had not learned to write readable code, and made limited progress learning to test their programs. College instructors complain that they, too, have difficulty teaching these aspects of programming. It may be that the emphasis on readability should be postponed until it is essential. When students start to write large programs that they themselves cannot keep track of they are likely to learn to write readable code. Testing, however, is viewed as important to students but not easily learned. Perhaps it requires more emphasis in the curriculum, or innovative teaching approaches.

The talented students were, however, by far superior programmers in comparison to other students in the ACCCEL sample.

Furthermore, their performance can be traced directly to the instructional provisions and emphases of their courses. One student in the subsample had gained most of his programming experience outside of classroom instruction and only 7% reported having had out-of-school instruction. Most of the subsample had limited or no formal experience with programming prior to enrolling in the course. It is safe to assume that these talented students entered the computer curriculum as virtual novices and thus their accomplishments can be attributed to the instruction provided in their courses.

Instructional emphases, curriculum materials, and teacher behavior were reflected in the students' performances, as indicated by the results of the discriminant analyses, performance trends, and observation reports. The discriminant analyses correctly classified students' performance by school. Independent raters who examined the programs in blind review also were able to determine the students' school affiliations and distinguish instructional emphases in the projects.

Site 2. Students in this self-paced curriculum generally were responsible for their own learning and progress. Most students were not sufficiently autonomous to gain from the course. The talented students, with the exception of the outstanding student, in particular, managed to overcome the lack of structure to some degree. On the whole, performance was mediocre when asked to write a program under time pressure. The students lacked the self-discipline needed to produce such a product. The programs were unreadable, particularly in listing form, and of average quality and structure. Under time pressure, the students failed to met the assignment's specifications, and attempted even fewer embellishments. That is, they put as little as possible into their programs. It was obvious that Site 2's students were deficient in their understanding of control structures. They were unable to design a functional subroutine or loop. However, performance was slightly better when given a longer assignment. The quality of their programs improved notably, as did their attempts to meet the program specifications and add embellishments.

In general, these students had gained only limited knowledge of programming and had not even mastered the topics of the textbook lessons that they had studied. They had knowledge of a variety of elementary language features, but were unable to apply more complex ones or integrate them into templates. The focus on language features reflects the instructional emphasis on the very basics of the programming language. Little or no attempt was made to teach the students procedural skills, more complex features of the language, or structured design techniques. Thus, it was apparent that students were deficient in their testing, debugging, and reformulating skills, in part because they never were taught these skills nor modeled by the teacher. Students at Site 2 were not only limited to what they had learned in their course, but actually were hindered in their progress to gain

additional knowledge about programming due to the structure of the curriculum. These students did not have the procedural skills required to program at even a low-level, nor did they have the generalizable problem solving skills that would assist them in further learning. The structure of the course discouraged autonomous learning, therefore the students failed to go beyond what they had been taught explicitly and failed to take advantage of the exploratory environment in which they worked.

Site 5. The instructional emphases at Site 5 were on making programs readable, structuring them properly (not necessarily efficiently, but making them look good), using templates, and following the textbook's instructional sequence. Programs produced by students at this school reflected these foci. Students produced programs that were highly readable in listing form. These programs were readily distinguishable from the work of students at other schools. The structure of the programs was distinctive; that is, all students at Site 5 were required to produce listings in a specified format. Thus, each program was formatted in an identical manner.

Programs generally were of average quality, showing some understanding of control structures and attempts to meet most of the assignment's specifications. Surprisingly, students at Site 5 attempted to add very few embellishments to their work, on either the On-line or Long-Term Programming Assessments. In part, this lack of embellishment reflects the teacher's directive that students were not to include in their programs any procedure or feature that had not been covered in class. Students were prohibited from going beyond the scope of instruction, even if they were familiar with a particular programming concept. Thus, when given the opportunity to show their expertise, students were unwilling or unable to accept the challenges engendered in the embellishments. Again, this is an unfortunate finding; students failed to explore an environment which encourages such autonomous and discovery learning opportunities. This finding was noted for performance on the BASIC tasks. However, the students were somewhat more adept in their exploration of the Spider World simulation. They readily adapted to the new environment and made good progress in their one-hour exposure to the program. Thus, these students were able to take advantage of the computer learning environment once the content was separate from their previous instruction. Yet, when given the chance to experiment with a familiar programming language, they failed to go beyond what they had been taught.

Site 6. Students at Site 6 produced the most efficient and best designed programs of all the schools. Reflecting an instructional emphasis, their programs were the only ones that were sufficiently documented to stand alone. That is, the students included directions and features to make the programs readable on-line. The programs, however, were less readable in listing form; that is, they often failed to include comments to assist the reader's understanding of the procedures. The students

produced programs of good quality that included most of the assignment's specifications. Their knowledge of looping, subroutines, and control structures was apparent in the programs (also an instructional emphasis). Students at Site 6 took some risks with their programs, attempting complicated embellishments that would enhance their work and make them more interesting.

In general, students at Site 6 had progressed farthest along the chain of cognitive accomplishments. They not only had mastered the simple and complex language features of BASIC, but more importantly had acquired the inquiry and autonomous learning skills needed to become competent programmers. Evidence for this finding was observed in both their performance on the BASIC tasks as well as in Spider World. They adeptly explored the new environment, tested its limitations, and acquired an understanding of its basic features from only a brief introduction. Moreover, these students differed from those at the other sites in that they were not limited by the instruction they received. Rather, the teacher encouraged students to go beyond what they were taught, explore the environment, and acquire both procedural and declarative knowledge independently.

The teacher's emphasis was on general problem solving skills. This focus was apparent in the way students tested, debugged, and reformulated their programs. They systematically identified errors in their work and determined the most effective way to remediate the bug. They efficiently planned and structured their work, also reflecting one of the teacher's emphases. Thus, the students at Site 6 had not only acquired sufficient content knowledge about programming, but also were making excellent progress in developing the all-important procedural and generalizable problem solving skills.

Overview. In conclusion, the talented students were by far superior programmers to the other students in the ACCCEL sample. This select subsample had made progress in acquiring the content knowledge necessary to become good programmers. They were less advanced, however, in their procedural skills. They were able to design reasonably complex programs, but were deficient in testing, debugging, and reformulating. Thus, they were at the design link in the chain of cognitive accomplishments. Interestingly, they exhibited good inquiry and problem solving skills when introduced to a novel computer learning environment, thereby showing some evidence of generalizable problem solving skill.

Instruction was an important influence on the knowledge and skills displayed by the talented students. Distinct instructional emphases were readily apparent in their work. These students had acquired some programming skills and knowledge in their classes, but many failed to go beyond the boundaries of their instruction to explore the computer learning environment. On the whole, the majority of the talented middle school programmers appeared to have gained their talent from good classroom instruction. However, in order to become more skilled, these students will need

to take advantage of the discovery learning opportunities provided by the environment.

References

Ali, A. M. (1981). The use of positive and negative examples during instruction: Some important issues related to the design and development of instructional materials. Journal of Instructional Development, 5, 2-7.

Anderson, R. C. (1967). Educational psychology. Annual Review of Psychology, 18, 129-163.

Atwood, M. E., & Ramsey, H. R. (1978). Cognitive structures in the comprehension and memory of computer programs: An investigation of computer debugging (Tech. Rep. TR-78-A210). Alexandria, VA: U. S. Army Research Institute for the Behavioral and Social Sciences.

Cronbach, L. J., & Snow, R. E. (1977). Aptitudes and instructional methods: A handbook for research on interactions. New York: Irvington Publishers.

Dalbey, J., & Linn, M. C. (in press a). Cognitive consequences of programming: Augmentations to BASIC instruction. Journal of Educational Computing Research.

Dalbey, J., & Linn, M. C. (in press b). The demands and requirements of computer programming: A review of the literature. Journal of Educational Computing Research.

Doyle, W. (1983). Academic work. Review of Educational Research, 53, 159-199.

Egan, D. E., & Greeno, J. G. (1973). Acquiring cognitive structure by discovery and rule learning. Journal of Educational Psychology, 64, 85-97.

Hayes-Roth, F. (1978). Learning by example. In A. M. Lesgold, J. W. Pellegrino, S. D. Fokkema, & R. Glaser (Eds.), Cognitive psychology and instruction. New York: Plenum.

Hayes-Roth, F., & McDermott, J. (1976). Learning structured patterns from examples. Proceeding of the Third International Joint Conference on Pattern Recognition.

Kurland, D. M., Mawby, R., & Cahir, N. (1984). The development of programming expertise in adults and children. Paper presented at the annual meeting of the American Educational Research Association, New Orleans.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. Educational Researcher, 14(5), 14-16, 25-29.

Linn, M. C., & Dalbey, J. (in press). Cognitive consequences of programming instruction: Influence of instructional setting, student access, and student ability. Educational Psychologist.

Linn, M. C., Fisher, C. W., Dalbey, J., Mandinach, E. B., Burbules, N. C., & Reese, P. (1983). The effects of learning to program and learning from cognitively demanding software: Research design (ACCCEL Tech. Rep. No. 8). Berkeley, CA: University of California, Lawrence Hall of Science.

Lockheed, M. C., & Mandinach, E. B. (1985). Cognitive consequences of computer applications. Princeton, NJ: Educational Testing Service.

Luehrmann, A., & Peckham, H. (1983). Computer literacy: A hands-on approach. New York: McGraw-Hill.

Mandinach, E. B. (1984). The role of strategic planning knowledge and self-regulation in learning an intellectual computer game. Unpublished doctoral dissertation, Stanford University, Stanford, CA.

Mandinach, E. B. (in press). Clarifying the "A" in CAI for learners of different abilities. Journal of Educational Computing Research.

Mandinach, E. B., & Fisher, C. W. (1984). Review of research on the cognitive effects of computer-assisted learning (ACCCEL Tech. Rep. No. 4). Berkeley, CA: University of California, Lawrence Hall of Science.

Mandinach, E. B., & Fisher, C. W. (1985). Individual differences and acquisition of computer programming skill (ACCCEL Tech. Rep. No. 30). Berkeley, CA: University of California, Lawrence Hall of Science.

Mandinach, E. B., & Linn, M. C. (1985). Cognitive consequences of programming: Achievements of experienced and talented programmers (ACCCEL Tech. Rep. No. 26). Berkeley, CA: University of California, Lawrence Hall of Science.

Mandinach, E. B., & Linn, M. C. (in press). Cognitive consequences of programming: Achievements of experienced and talented programmers. Journal of Educational Computing Research.

Mayer, R. E., & Greeno, J. G. (1972). Structural differences between learning outcomes produced by different instructional methods. Journal of Educational Psychology, 63, 165-173.

Shulman, L. S., & Keislar, E. R. (Eds.). (1966). Learning by discovery: A critical appraisal. Chicago: Rand McNally.

Snow, R. E. (1977). Research on aptitude for learning: A progress report. In L. S. Shulman (Ed.), Review of research in education (Vol. 4). Itasca, IL: Peacock.

Stein, J. S., & Linn, M. C. (1985). Capitalizing on computer-based interactive feedback: An investigation of Rocky's Boots. In M. Chen & W. Paisley (Eds.), Children and microcomputers: Formative studies. Beverly Hills: Sage.

Tobias, S. (1982). When do instructional methods make a difference? Educational Researcher, 11(4), 4-9.

Footnote

This study was funded by the National Institute of Education under Grant No. 400-83-0017. Any opinions, findings, and conclusions, or recommendations expressed in the paper do not necessarily reflect the views of the National Institute of Education. The author wishes to thank the members of the ACCCEL Project, John Dalbey, Charles Fisher, Marcia Linn, Velda Mark, Corrine Nelson, and Joanne Stein for their contributions to the study.

Table 1

Selected Site Characteristics

Variable	Site 1	Site 2	Site 3	Site 4	Site 5	Site 6/6a
School Enrollment	943	930	649	645	634	453
ACCCEL Substudy[1]	Programming Introductions	Programming Introductions, Experienced Programmers	Programming Introductions	ACCCEL Explicitness	Naturally Occurring Explicitness, Experienced Programmers	Naturally Occurring Explicitness, Experienced Programmers
CLASSROOM CONTEXT						
Number of Classes in ACCCEL Studies	3	3	6	3	2	2
Approximate Course Duration	18 weeks 3.5 days/wk.	12 weeks 5 days/wk.	12 weeks 5 days/wk.	32 weeks 5 days/wk.	32 weeks 5 days/wk.	12 weeks 5 days/wk.
Available On-line Time During Class (per person in minutes per day)	15	20	10	20	10	20
Other School Access (outside of class time)	Occasional	Occasional	Lunch (45 min./day)	Occasional	2 hours/ week	15 hours/ week
Number and Kind of Computers	8 Apples	12 Apples	8 Apples	11 Pets	9 Apples	5 Apples 3 Pets 2 Olivettis
STUDENT INFORMATION						
Number of Students	92	84	132	51	60	50
Grade	8	7/8	7/8	7/8	8	8
Selection Criteria	Math/Science Class Required	Elective; No Prerequisite	Elective; No Prerequisite	Elective; No Prerequisite	Elective; No Prerequisite	Elective; B+ or Higher Average in English and Algebra; Further Screening by Teacher
TEACHER INFORMATION						
Number of Years Taught Programming	0	2	1.5	2.5	2	More than 3
Area of Specialization	Math/ Science	Math	Industrial Education	Special Education	Gifted Coordinator	Math
Computer Training	Self-taught	Self-taught, Seminars, and Conferences	Self-taught, Seminars	Self-taught	Self-taught, Seminars	Formal Computer Education
CURRICULUM						
Format	Self-paced, Informal	Self-paced	Daily Assignments, Frequent Tests	Daily Program Based on Teacher Approval	Daily Assignments, Optional Projects	Projects with Timelines
Instructional Materials	Kids & the Apple	Computer Literacy (Luehrmann); Own Support Materials	Kids & the Apple; Other Support Materials	Teacher-selected materials	Computer Literacy (Luehrmann); Own Support Materials	Teacher-selected Materials
Estimated Curriculum Difficulty	Moderate/Low	Moderate	Low	Moderate/High	Moderate/High	High

[1]Students from all six sites are included in the individual differences substudy.

Table 2

Comparison of all ACCCEL Sites, ACCCEL Exemplary Sites Experienced Programmers, and Talented Programmers

Variables	ACCCEL[a] Sites	Exemplary[b] Sites	Experienced[c]	Talented[d]
		Final Programming Assessment (%)		
Comprehension	45.8	79.3	85.0	92.9
Reformulation	40.6	68.7	68.9	88.0
Design	28.6	65.6	75.3	88.2
Total	38.3	71.9	77.8	90.1
		Ability		
Raven	8.27	9.76	9.67	10.27
Vocabulary	31.20	42.16	44.82	42.64
		Computer Experience		
Home Computers (%)	.43	.50	.62	.71
Out of School Instruction (%)	.16	.16	.15	.07
		Gender		
Percent Males	.64	.58	.60	.38
Percent Females	.36	.42	.40	.62

Note. [a]n = 525. [b]n = 110. [c]Experienced students are those identified as potentially talented but were not selected for the talented group, n = 71. [d]Talented students are the top programmers from all ACCCEL sites, n = 24.

Table 3

Performance of all Talented Programmers on the On-Line Programming Assessment (n=22)

Variable	M	% of Maximum	Minimum	Median	Maximum
On-line Readability	4.4	44	1	3.8	10
Listing Readability	5.8	58	2	6.5	10
Control Structures	13.4	67	0	17.9	20
Quality	14.0	70	5	14.9	20
Specifications	15.3	61	0	19.2	25
Embellishments	13.8	21	0	10.0	65

Table 4

Performance of all Talented Programmers on the On-Line Programming Assessment by School (n=22)

Variable	School					
	School 2[a]		School 5[b]		School 6[c]	
	M	%	M	%	M	%
On-line Readability	5.0	50	3.4	34	5.7	57
Listing Readability	3.2	32	7.9	79	4.0	40
Control Structures	6.2	31	14.5	73	15.7	78
Quality	11.2	56	13.4	67	16.4	82
Specifications	11.2	45	15.2	61	17.8	71
Embellishments	16.2	25	11.7	18	15.7	24

Note. [a] n = 4. [b] n = 11. [c] n = 7.

Table 5

Talented Programmers' Performance on the Long-Term Programming Assessment (n=24)

Variable	M	% of Maximum	Minimum	Median	Maximum
On-line Readability	5.8	58	1	5.2	10
Listing Readability	6.9	69	3	7.5	10
Control Structures	13.1	66	0	12.5	20
Quality	12.4	62	5	11.8	20
Specifications[a]	76.2	76	14	86.5	105
Embellishments	20.6	28	0	15.0	75

Note. [a] The specifications variable is reported as a percent of possible points. A bonus of 5 points was added to the score if boundary conditions were considered. Maximum possible score was 105.

Table 6

Talented Programmers' Performance on the Long-Term Programming Assessment by School and Program (n=24)

Variable	School					
	School 2[a]		School 5[b]		School 6[c]	
	M	%	M	%	M	%
On-line Readability	3.0	30	5.3	53	7.6	76
Listing Readability	3.0	30	7.4	74	8.0	80
Control Structures	8.8	44	13.6	68	14.4	72
Quality	13.8	64	12.2	61	12.1	60
Specifications	87.8	88	67.6	68	81.6	82
Embellishments	26.2	35	13.6	18	26.7	36

Note. [a] n = 4. [b] n = 11. [c] n = 9.